Класс SystemAuthentication

Редакция от 13.10.2026 · docs.kb-agava.ru

Содержание

1 Получение системы аутентификации 2
2 Перечисление Privilege 2
3 Методы 2
  3.1 GetCurrentUser() 2
  3.2 GetRole() и GetUser() 3
  3.3 HasAccess() 3
  3.4 AddPrivilege() 4
  3.5 RemovePrivilege() 5
  3.6 AddUserRole() и RemoveUserRole() 5
  3.7 Ошибки аргументов и результат изменения прав 6
4 Правила проверки доступа 6
5 Глобальные функции аутентификации 7
  5.1 GetCurrentUser() и UserInfo 7
  5.2 HasAccess() по пути 7
  5.3 ShowLoginDialog() 8
  5.4 Logout() 8
6 AuthUser_t и AuthRole_t 8
7 Общий интерфейс узлов 8

Класс `SystemAuth` представляет узел `Система аутентификации`.

Описание соответствует скриптовому API текущей версии `AS 1.7.8+`.

Общие методы узла соответствуют BasicNode.

1 Получение системы аутентификации

```
SystemAuthentication_t@ GetSystemAuthentication()
```

Глобальная функция возвращает систему аутентификации локальной станции. Если такой системы нет, результат — `null`. Функция не создаёт новый узел.

```
string GetAuthenticationSystemName()
{
    SystemAuthentication_t@ auth = GetSystemAuthentication();

    if (auth is null)
    {
        return "";
    }

    return auth.GetName();
}
```

Необходимо проверять результат на `null` перед обращением к методам.

2 Перечисление Privilege

Значение	Число	Назначение
`Privilege::ReadValue`	1	Чтение значения узла.
`Privilege::WriteValue`	2	Запись значения узла.
`Privilege::OpenWindow`	3	Открытие окна.
`Privilege::CloseWindow`	4	Закрытие окна.

В скрипте следует использовать именованные значения перечисления. Произвольные числовые приведения, например `Privilege(42)`, не являются поддерживаемыми привилегиями.

3 Методы

3.1 GetCurrentUser()

```
AuthUser_t@ GetCurrentUser()
```

Возвращает ссылку на узел текущего авторизованного пользователя. Если текущий пользователь отсутствует или ссылка на него не разрешается в узел пользователя, результат — `null`.

Это **живой узел проекта**, а не снимок пользовательских данных. После переключения пользователя ранее полученная ссылка по-прежнему указывает на прежний узел; для получения нового текущего пользователя нужно вызвать метод повторно.

```
string GetCurrentUserNodeName()
{
    SystemAuthentication_t@ auth = GetSystemAuthentication();

    if (auth is null)
    {
        return "";
    }

    AuthUser_t@ user = auth.GetCurrentUser();

    if (user is null)
    {
        return "";
    }

    return user.GetName();
}
```

3.2 GetRole() и GetUser()

```
AuthRole_t@ GetRole(const string &in name)
AuthUser_t@ GetUser(const string &in name)
```

Ищут существующую роль или пользователя в основной коллекции узлов проекта. Параметр `name` сравнивается с именем узла `GetName()` точно, с учётом регистра. Это **не путь**, не шифр `GetShortName()` и не описание `GetDescription()`.

При отсутствии совпадения возвращается `null`. При нескольких совпадениях возвращается первый найденный узел соответствующего типа; для однозначного поиска рекомендуется давать ролям и пользователям уникальные имена.

3.3 HasAccess()

```
bool HasAccess(BasicNode_t@+ node, Privilege privilege)
```

Проверяет указанную привилегию **текущего пользователя** для заданного узла. Возвращает `true` для разрешения, заданного непосредственно на узле или унаследованного от его предка. Отсутствие разрешения, запрет и отсутствие текущего пользователя дают `false`.

Для прокси, созданного по пути к узлу проекта, проверяется исходный узел проекта; для узла, созданного скриптом, — сам этот узел. Прокси `BasicNode` и его потомки можно передавать благодаря неявному приведению к `BasicNode_t@`.

Передача `null` или недопустимого значения `Privilege` вызывает исключение скрипта. Проверка не выполняет вход пользователя и не показывает диалог аутентификации.

```
bool CanReadNode(BasicNode@ node)
{
    if (node is null)
    {
```

```
        return false;
    }

    SystemAuthentication_t@ auth = GetSystemAuthentication();

    if (auth is null)
    {
        return false; // Политика данного примера: отсутствие системы — отказ.
    }

    return auth.HasAccess(node, Privilege::ReadValue);
}
```

Возврат `false` при отсутствии системы в этом примере — решение самого скрипта. Поведение глобальной функции `HasAccess()` отличается и описано ниже.

3.4 AddPrivilege()

```
bool AddPrivilege(AuthRole_t@+ role, BasicNode_t@+ node, Privilege privilege, bool allow =
true)
```

Устанавливает правило указанной привилегии для роли и узла:

- `allow = true` — разрешение;
- `allow = false` — явный запрет.

Прежние непосредственные записи той же привилегии для этой роли и узла заменяются новым правилом. Повторный вызов не накапливает дубликаты. При корректных аргументах результат — `true`, в том числе при повторной установке уже заданного правила.

Правило относится к роли, а не исключительно к текущему пользователю, поэтому оно может влиять на всех пользователей этой роли. Метод сам по себе не назначает роль пользователю и не выполняет вход.

```
bool SetOperatorReadAccess(BasicNode@ node, bool allow)
{
    if (node is null)
    {
        return false;
    }

    SystemAuthentication_t@ auth = GetSystemAuthentication();

    if (auth is null)
    {
        return false;
    }

    AuthRole_t@ role = auth.GetRole("Operator");

    if (role is null)
    {
        return false;
    }

    return auth.AddPrivilege(role, node, Privilege::ReadValue, allow);
}
```

Имя `Operator` — пример; следует использовать имя роли своего проекта. Для узлов, созданных по пути, правило устанавливается на исходный узел проекта. Права, выданные через этот API на узел, удаляются из ролей при уничтожении этого узла.

3.5 RemovePrivilege()

```
bool RemovePrivilege(AuthRole_t@+ role, BasicNode_t@+ node, Privilege privilege)
```

Удаляет непосредственные записи указанной привилегии для роли и узла — как разрешения, так и запреты. Возвращает `true`, если была удалена хотя бы одна запись, иначе `false`.

Удаление правила не равно запрету. Оно не удаляет правила на предках узла и не меняет другие роли пользователя. После удаления может снова действовать унаследованное разрешение или другое применимое правило. Для явного запрета используется `AddPrivilege(role, node, privilege, false)`.

```
bool RemoveOperatorReadRule(BasicNode@ node)
{
    if (node is null)
    {
        return false;
    }

    SystemAuthentication_t@ auth = GetSystemAuthentication();

    if (auth is null)
    {
        return false;
    }

    AuthRole_t@ role = auth.GetRole("Operator");

    if (role is null)
    {
        return false;
    }

    return auth.RemovePrivilege(role, node, Privilege::ReadValue);
}
```

3.6 AddUserRole() и RemoveUserRole()

```
bool AddUserRole(AuthUser_t@+ user, AuthRole_t@+ role)
bool RemoveUserRole(AuthUser_t@+ user, AuthRole_t@+ role)
```

`AddUserRole()` назначает пользователю существующую роль. Возвращает `true`, если роль добавлена, и `false`, если она уже была назначена.

`RemoveUserRole()` удаляет назначение роли. Возвращает `true`, если назначение удалено, и `false`, если такой роли у пользователя не было. Сам узел роли не удаляется, назначения этой роли другим пользователям не изменяются.

```
bool ChangeUserRole(const string &in userName, const string &in roleName, bool assign)
{
    SystemAuthentication_t@ auth = GetSystemAuthentication();
```

```
    if (auth is null)
    {
        return false;
    }

    AuthUser_t@ user = auth.GetUser(userName);
    AuthRole_t@ role = auth.GetRole(roleName);

    if (user is null || role is null)
    {
        return false;
    }

    if (assign)
    {
        return auth.AddUserRole(user, role);
    }

    return auth.RemoveUserRole(user, role);
}
```

В примере `false` может означать отсутствие изменения, а не ошибку: роль уже назначена либо уже отсутствует. Назначение роли не делает пользователя текущим и не выполняет проверку его пароля.

3.7 Ошибки аргументов и результат изменения прав

В объектных методах `HasAccess()`, `AddPrivilege()` и `RemovePrivilege()` недопустимая привилегия либо `null` вместо обязательного узла вызывает исключение скрипта. В методах изменения прав и ролей `null` вместо обязательного пользователя или роли также вызывает исключение. Эти случаи нельзя трактовать как обычный результат `false`: выполнение контекста прерывается исключением.

Модификатор `@+` в сигнатурах относится к управлению временем жизни аргументов; при вызове дополнительных действий по освобождению ссылок не требуется.

Изменения применяются к состоянию работающего рантайма. Обновление доступности виджетов выполняется для изменений прав на узлы проекта и для фактических изменений назначений ролей. Сам вызов метода не сохраняет проект на диск; сохранение настроек или повторное применение правил при запуске нужно организовать отдельно.

Методы изменения прав **не содержат отдельной проверки «текущий пользователь — администратор»**. Ограничения на запуск таких скриптов и изменение настроек следует обеспечивать в приложении. Результат `true` подтверждает выполнение операции, а не гарантирует итоговое разрешение доступа конкретному пользователю.

4 Правила проверки доступа

Правила роли могут распространяться на дочерние узлы. Внутри роли более близкое к проверяемому узлу правило проверяется раньше правила его предка.

При нескольких ролях текущая реализация перебирает роли в порядке их назначения и использует результат **первой роли, для которой найдено применимое правило** — разрешение или запрет. Это не объединение всех разрешений и не правило «любой запрет всегда важнее любого разрешения» для всех ролей сразу.

Не следует полагаться на противоречащие правила нескольких ролей без проверки результата `HasAccess()`. После удаления непосредственного правила может измениться то, какое унаследованное правило или какая роль определяет результат.

5 Глобальные функции аутентификации

5.1 GetCurrentUser() и UserInfo

```
const UserInfo@ GetCurrentUser()
```

Глобальная функция возвращает **снимок** данных текущего пользователя, либо `null`, если система аутентификации или текущий пользователь отсутствуют. Она отличается от метода `SystemAuthentication_t.GetCurrentUser()`, возвращающего живой узел `AuthUser_t@`.

`UserInfo` предоставляет методы:

```
string GetNodePath() const
string GetName() const
string GetFullName() const
```

`GetFullName()` возвращает описание узла пользователя. Снимок не изменяется при последующем переключении пользователя; нужно получить новый объект через глобальную функцию. Конструктор `UserInfo` из скрипта недоступен.

```
string GetCurrentUserCaption()
{
    const UserInfo@ user = GetCurrentUser();

    if (user is null)
    {
        return "";
    }

    return user.GetFullName();
}
```

5.2 HasAccess() по пути

```
bool HasAccess(string nodePath, Privilege privilege)
```

Проверяет права текущего пользователя на узел, найденный по пути. Если путь не разрешается ровно в один узел, результат — `false`. Если узел найден, но система аутентификации отсутствует, результат — `true`. При наличии системы проверяются права текущего пользователя; отсутствие пользователя даёт `false`.

В отличие от объектного метода аргументом является строка пути, а не `BasicNode_t@`. Проверка не показывает диалог входа и не выдаёт привилегию автоматически.

```
bool CanReadSetpoint()
{
    return HasAccess("/Project/Station/Signals/Setpoint", Privilege::ReadValue);
}
```

5.3 ShowLoginDialog()

```
void ShowLoginDialog(string predefinedUserPath = "")
```

Запрашивает отображение диалога аутентификации. Необязательный аргумент — **путь к узлу пользователя**, который должен быть предварительно выбран в диалоге; это не имя роли, не пароль и не автоматический вход.

Если системы аутентификации нет либо не подключён обработчик пользовательского интерфейса, функция ничего не делает. При наличии системы и обработчика UI неверный или неоднозначный путь, а также путь к узлу другого типа вызывают исключение скрипта.

Функция возвращает `void`. Результат входа проверяют повторным чтением текущего пользователя. Вызов обработчика диалога синхронный на вызывающем потоке; доставку в UI-поток при необходимости выполняет интерфейс.

```
void RequestOperatorLogin()
{
    ShowLoginDialog("/Project/Station/SystemAuth/Users/operator");
}
```

Для диалога без заранее выбранного пользователя вызывается `ShowLoginDialog()`.

5.4 Logout()

```
void Logout()
```

Глобальная функция выполняет выход. Если в проекте задан корректный пользователь по умолчанию, рантайм переключается на него **без запроса пароля**. В противном случае сбрасываются текущий пользователь и свойства его имени/описания. При отсутствии системы функция ничего не делает.

После `Logout()` результат `GetCurrentUser()` поэтому не обязательно равен `null`: текущим может стать пользователь по умолчанию.

6 AuthUser_t и AuthRole_t

Это ссылки на узлы `Пользователь` и `Роль` проекта. Получают их через `SystemAuthentication_t.GetUser()`, `GetRole()` и `GetCurrentUser()`. Типы допускают неявное приведение к `BasicNode_t@` и явное приведение из `BasicNode_t@` через `cast<AuthUser_t>`/ `cast<AuthRole_t>`.

В текущем скриптовом API у этих типов зарегистрированы общие методы узла. Специализированные операции назначения ролей и изменения прав вызывают у `SystemAuthentication_t`, а не как `user.AddRole()` или `role.AddPrivilege()`.

C++-методы `AuthUser.GetRoles()`, `PrivExists()` и `AuthRole.GetPrivs()` напрямую в скрипте не зарегистрированы.

7 Общий интерфейс узлов

Следующий интерфейс доступен у `SystemAuthentication_t`, `AuthUser_t` и `AuthRole_t`:

```
string GetName()
string GetShortName()
string GetDescription()
string GetPath()
void SetName(string name)
void SetShortName(string shortname)
void SetDescription(string description)
NodeProperty@ GetProperty(string &in name)
bool PropertyValueUpdated(string &in name)
Result Start()
Result Stop()
EventProperty@ OnPropertyUpdated
```

`OnPropertyUpdated` — свойство события изменения свойства узла. Обработчик имеет сигнатуру `void EventPropertyCB(NodeProperty@ prop)`. Подписка выполняется через `AddListener()`/`RemoveListener()` или `+=`/`-=`, как для других объектов `EventProperty`.

Отдельных событий входа/выхода пользователя в текущей регистрации `SystemAuthentication_t` нет. Не следует считать `OnPropertyUpdated` специализированным событием завершения аутентификации.

Жизненным циклом системы обычно управляет рантайм. Вызов `Start()` системы также выполняет аутентификацию пользователя по умолчанию; ручной запуск не требуется для обычного использования `GetSystemAuthentication()`.